Algoritmos y Estructura de Datos (AyED)
Práctica 01: Algoritmos de Ordenamiento

Estudiantes:
Franklin Canaza Ccori
Hayde Luzmila Humpire Cutipa
Jair Francesco Huaman Canqui
Jhoel Salomon Tapara Quispe

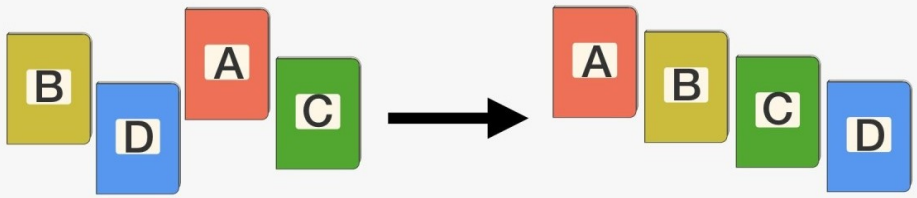
Docente: Dr. Vicente Machaca Arceda

Escuela Profesional de Ciencias de la Computación
Universidad Nacional de San Agustín

Contenido I

Introducción

Un algoritmo de ordenamiento se encarga de ubicar correctamente los elementos de un vector, ya sea en un orden ascendente o descendente. Los algoritmos de ordenamiento hacen uso de diversas técnicas, para lo cual un algoritmo de ordenamiento es eficiente en términos de tiempo y memoria, es por ello que se busca comparar sus resultados en el procesamiento de arreglos de bastantes elementos.



Requerimientos

Para realizar la comparación se han considerado los siguientes puntos:

- Se han tomado 5 arreglos para determinar 5 mediciones de la velocidad a una determinada cantidad de elementos, de esta manera calcular la velocidad promedio y las desviación estándar para calcular los límites superior o inferior.
- Se han generado arreglos con elementos aleatorios con dimensiones de 100, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000, 20000, 30000, 40000 y 50000.

Preparación de Datos

Para la generación de datos en archivos .txt se creó un script de nombre: generador.cpp con el lenguaje de programación C++, el cual genera números aleatorios y va escribiendo los mismo en un txt, genera 5 vectores de cada longitud. A continuación se muestra el código.

```
100
94712 25279 12526 51932 85616 67524 12415 90627 54038 56566 83175 16100 45800 80565 58117 58757 31158 20103 17497 92741 50073 19497 62204 94254 15285 69692 74481 77879 95583 99271
34638 90296 24550 42040 37105 5044 9564 49519 95670 58479 962 78846 74578 41638 54288 27572 395 85445 42551 12768 78186 87501 27141 35267 81755 42425 4959 51114 15182 543 45262 44696
90838 69811 86736 22820 74854 96299 67215 65402 49655 68176 39125 24234 4691 93412 46682 99962 73735 84110 12730 46798 71611 39871 82065 48243 77172 81900 94233 87230 82442 39495
31927 68157 4184 18663 90976 73914 9839 53068 34193 59494 21244 73317 78604 20812 61607 20163 20774 30219 4273 33503 77016 70760 68250 53958 13881 40300 35859 8114 22407 13178 47609
54333 81335 46669 72995 67188 20583 77711 20257 54776 37205 36377 22970 15809 57188 79453 30848 77961 9672 35121 6342 81565 758 69468 35523 14638 4645 66258 17629 27052 79436 60114
81384 60771 6783 49257 22836 27365 26968 43092 82140 59049 74346 5111 69734 31534 84563 582 4373 89112 30579 5591 70677 31337 69935 1077 40851 74580 62212 58479 96508 41648 13470
77892 97295 20252 22026 20131 47617 43870 58100 24634 2919 32446 24621 72652 58856 4062 68111 58105 93173 98689 63695 58727 24903 28508 54680 65754 97964 16892 19110 94472 53416 32580
67241 45588 47708 89267 60596 95324 33137 18696 14836 30933 46018 39456 98461 99750 43517 66572 57856 36690 65262 16428 90293 90164 44935 39851 50795 42899 56742 64782 32248 5036
97361 99489 50623 45069 88756 6096 35271 16770 19668 50106 42579 60562 84438 41041 60313 27956 7613 13046 59522 67751 29473 44693 52793 69285 84543 3588 7061 36162 68369 39309 41197
60607 38798 86697 554 22431 92792 30701 34077 7338 75683 71533 67899 60121 12574 28212 82953 15063 41257 42476 82814 65607 87168 30484 29769 66588 34071 36829 2750 97317 71014 43947
52801 4689 25521 53354 21996 13190 84054 56073 20527 54614 22483 88426 14736 35056 11515 97688 44995 47649 35041 27809 13256 17086 53169 43024 83673 87240 74729 86423 79434 45744
25247 32235 45309 50767 80466 67305 58833 64520 18255 79360 19135 40737 62663 28747 75792 69054 21312 20787 16703 56353 43473 29958 68315 91518 67858 51989 78758 42588 33289 53069
83208 58535 85304 28517 4179 65770 95821 63011 25167 8953 37248 39178 49689 94787 62801 20358 63842 84113 36022 80544 35343 74371 5380 3658 65889 73237 50523 39525 10702 83811 92593
93909 37223 72774 17304 36278 38544 8002 94166 58588 16955 31414 97765 66643 21079 60567 87001 84920 39557 17900 60341 74899 92270 65720 73433 58159 38958 23957 97683 44536 2645 85154
38446 39868 57928 50626 76145 96472 58627 65188 49937 75581 91479 47702 37102 12558 3146 18980 92354 42702 36879 52695 12478 24026 18416 85911 82184 52250 9868 74745 96785 12512 59899
30108 47256 17827 80733 18279 9176 34238 83466 59112 4696 74945 1692 41797 82379 99713 55653 74733 42416 87408 27429 49770 11434 40721 35681 88495 92970 40425 63240 84632 47814 18016
14741 89946 30719 90350 8225 34772 24588 86568
```

Figura: Resultado: Archivo Puntos.txt

Algoritmos

Se han implementado 4 algoritmos, los cuales tienen las siguientes complejidades en tiempo:

1. *Counting Sort* (Conteo): Complejidad $O(n + k)$
2. *Quick Sort* (Rápido): Complejidad $O(nlogn)$
3. *Insertion Sort* (Inserción): Complejidad $O(n^2)$
4. *Heap Sort* (Montones): Complejidad $O(nlogn)$

Counting Sort: Complejidad

Counting Sort	Mejor Caso	Caso Promedio	Peor Caso
Complejidad	O(n+k)	O(n+k)	O(n+k)

Cuadro: Complejidad - *Counting Sort*

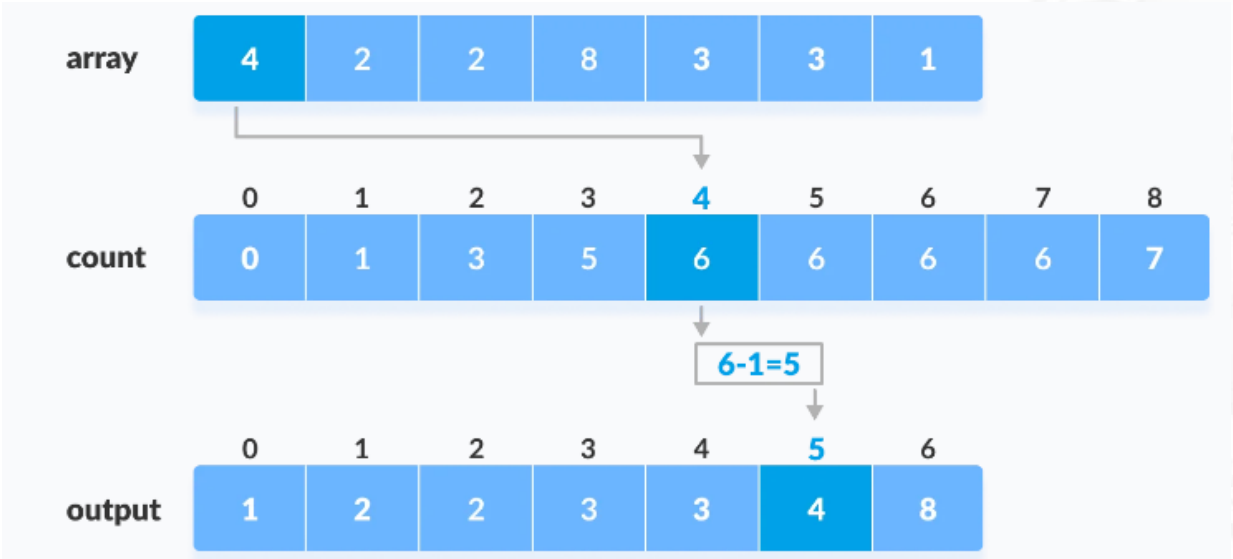


Figura: Ejemplo - *Counting Sort*

Counting Sort: Resultados

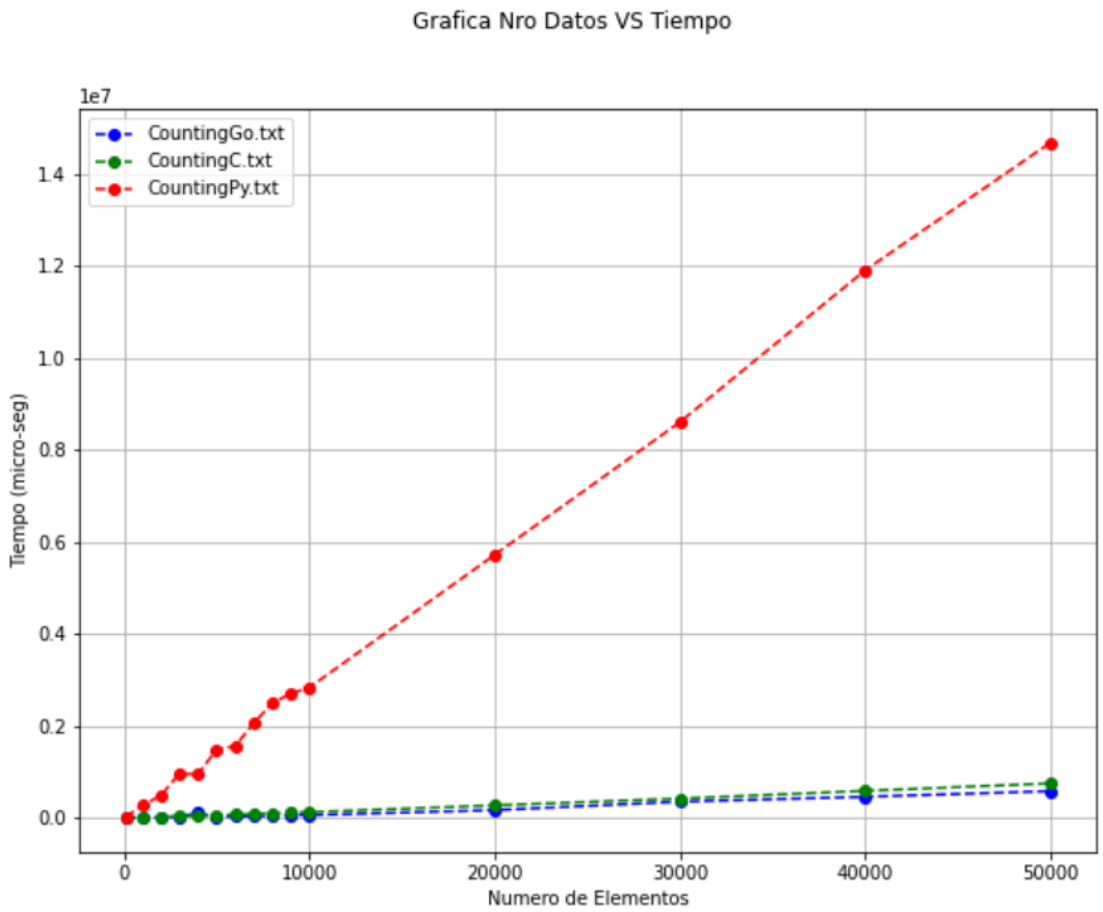


Figura: Comparación de *Counting Sort*

Counting Sort: Limite Superior y Limite Inferior

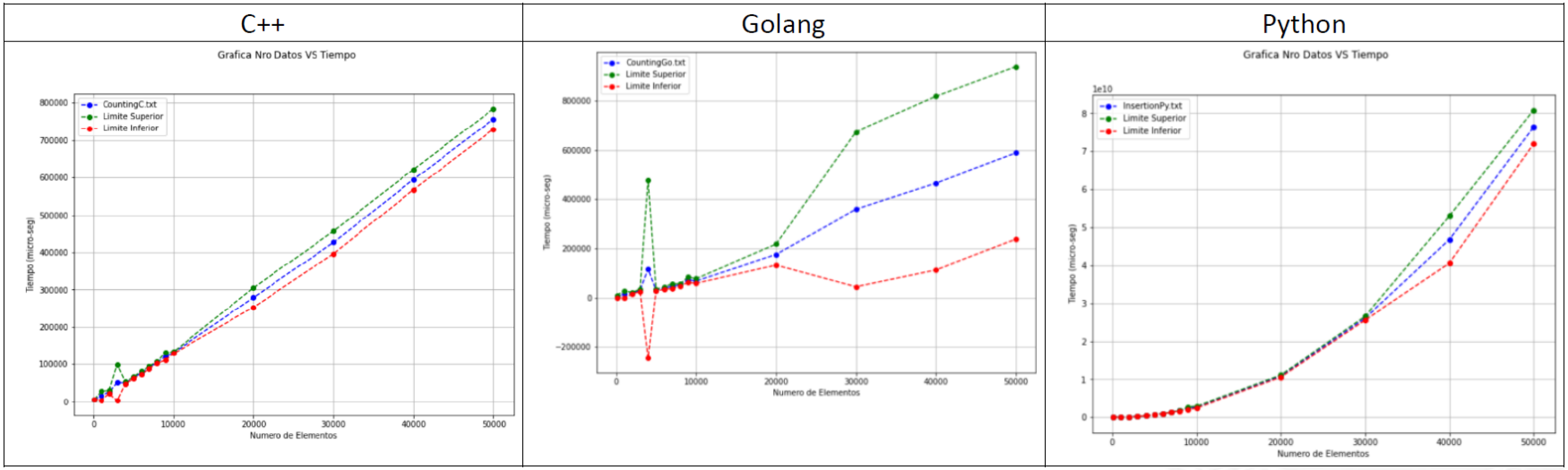


Figura: *Counting Sort*: Limite Superior y Limite Inferior

Quick Sort: Complejidad

Quick Sort	Mejor Caso	Caso Promedio	Peor Caso
Complejidad	O(nlogn)	O(nlogn)	$O(n^2)$

Cuadro: Complejidad: *Quick Sort*.

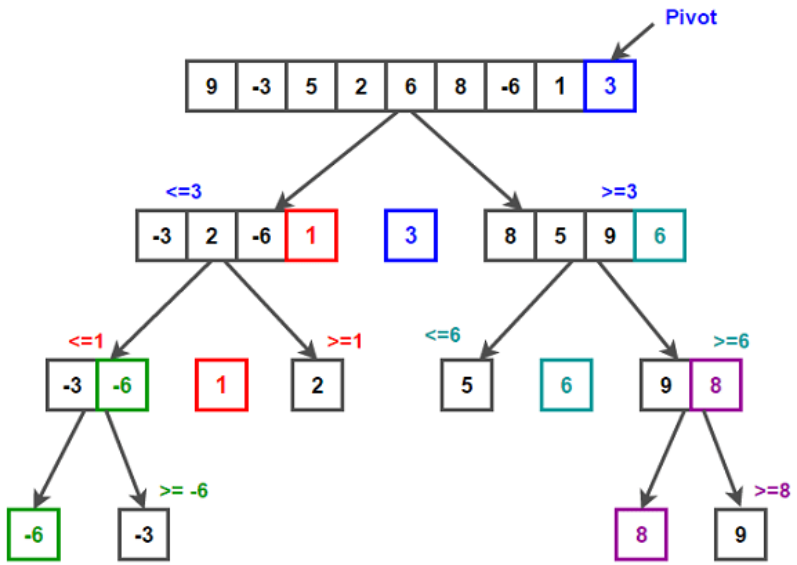


Figura: Ejemplo: *Quick Sort*

Quick Sort: Resultados

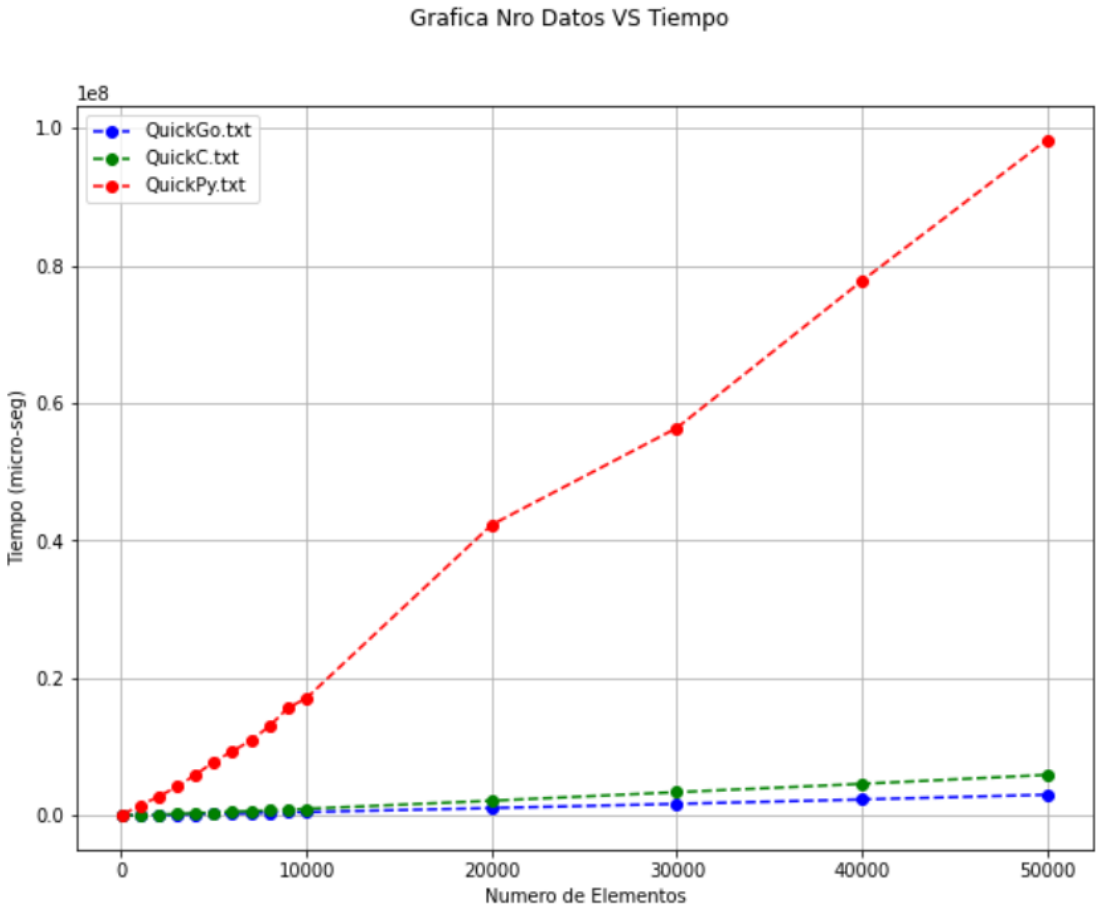


Figura: Comparación de *Quick Sort*

Quick Sort: Limite Superior y Limite Inferior

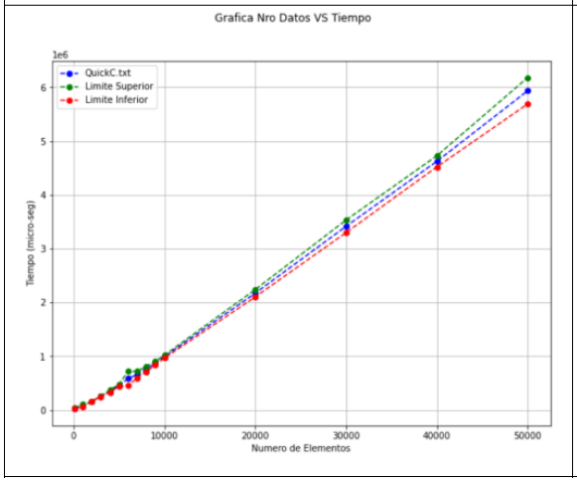
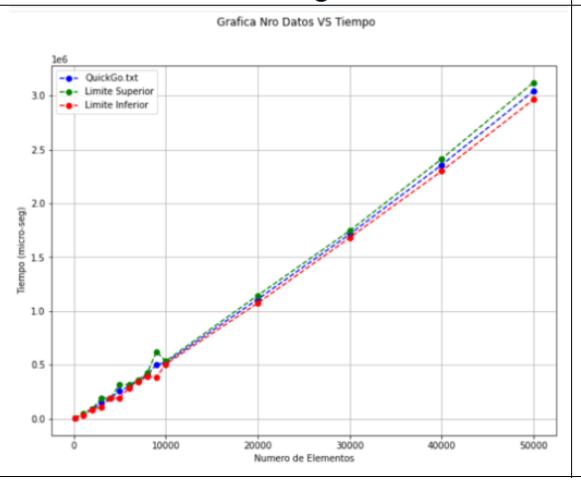
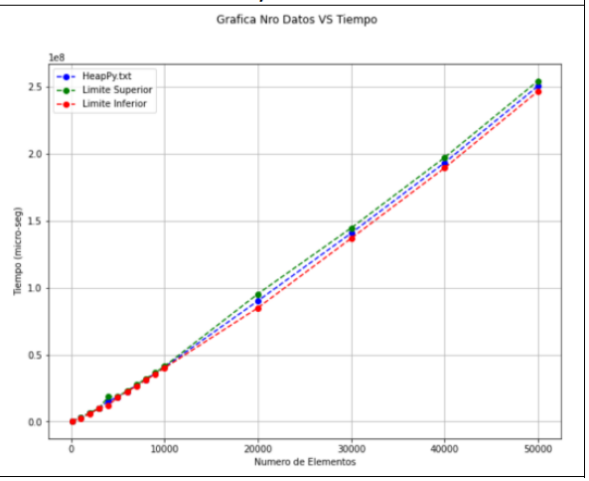
C++	Golang	Python
		

Figura: *Quick Sort*: Limite Superior y Limite Inferior

Insertion Sort: Complejidad

Insertion Sort	Mejor Caso	Caso Promedio	Peor Caso
Complejidad	O(n)	O(n^2)	O(n^2)

Cuadro: Complejidad: Insertion Sort

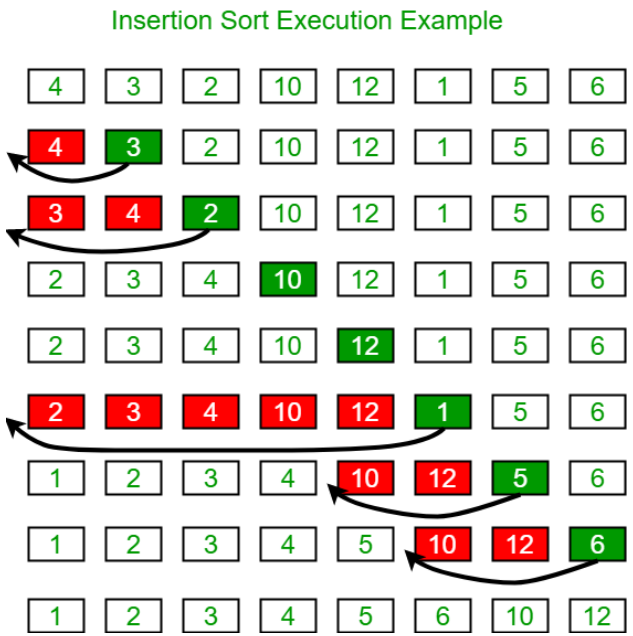


Figura: Ejemplo: Insertion Sort

Insertion Sort: Resultados

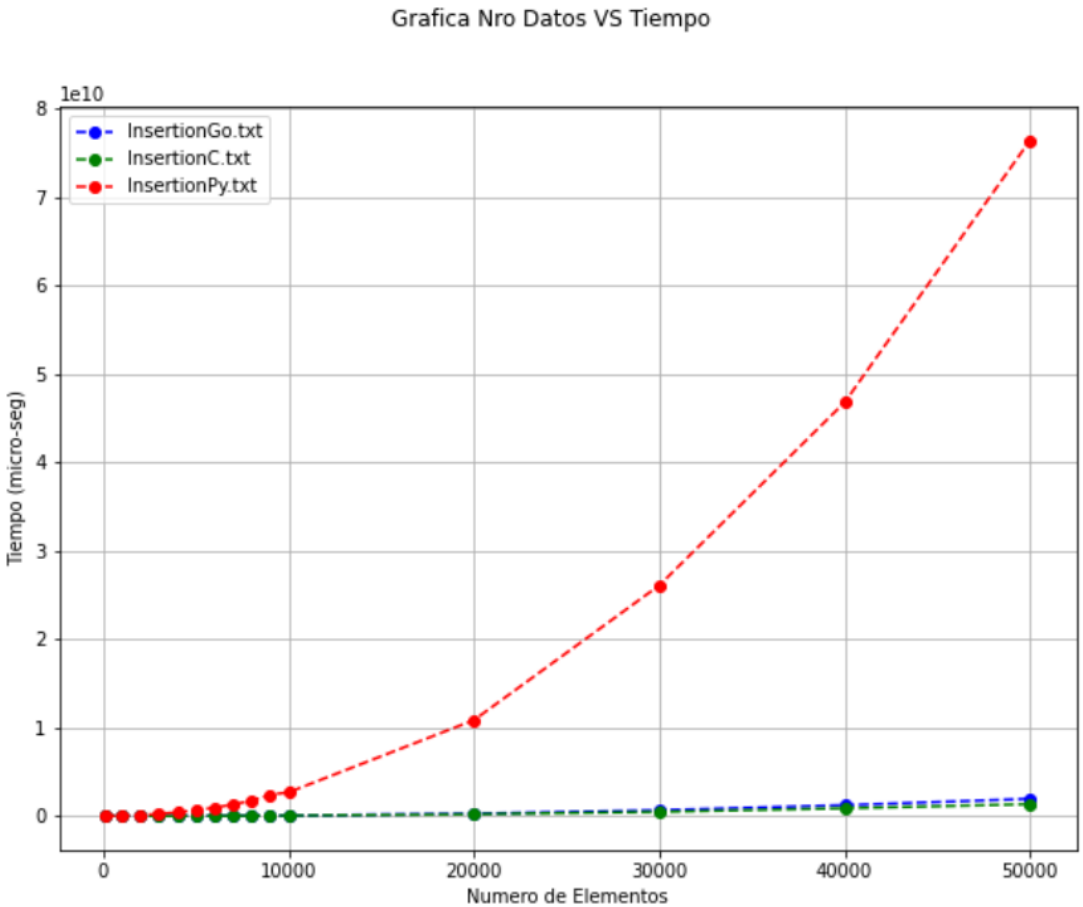


Figura: Comparación de *Insertion Sort*

Insertion Sort: Limite Superior y Limite Inferior

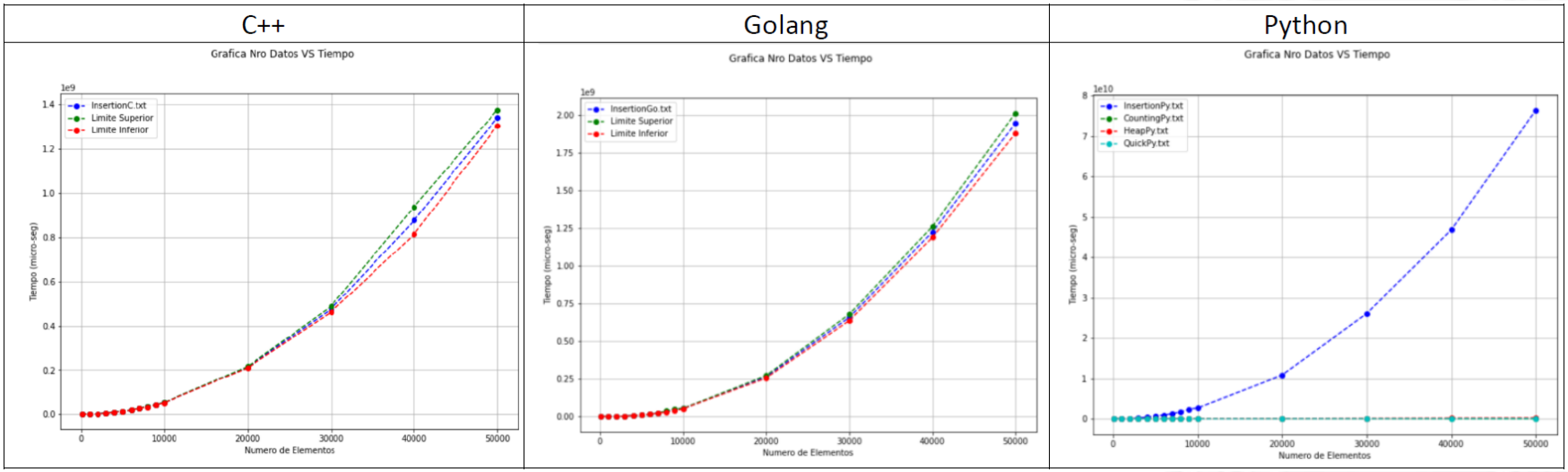


Figura: *Insertion Sort*: Limite Superior y Limite Inferior

Heap Sort: Complejidad

Heap Sort	Mejor Caso	Caso Promedio	Peor Caso
Complejidad	O(nlogn)	O(nlogn)	O(nlogn)

Cuadro: Complejidad: Heap Sort.

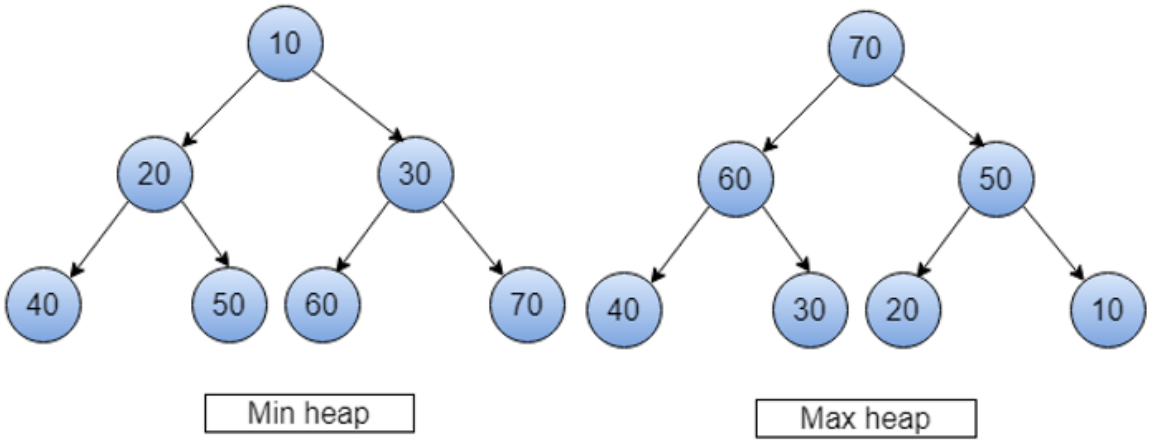


Figura: Ejemplo: Max Heap y Min Heap

Heap Sort: Resultados

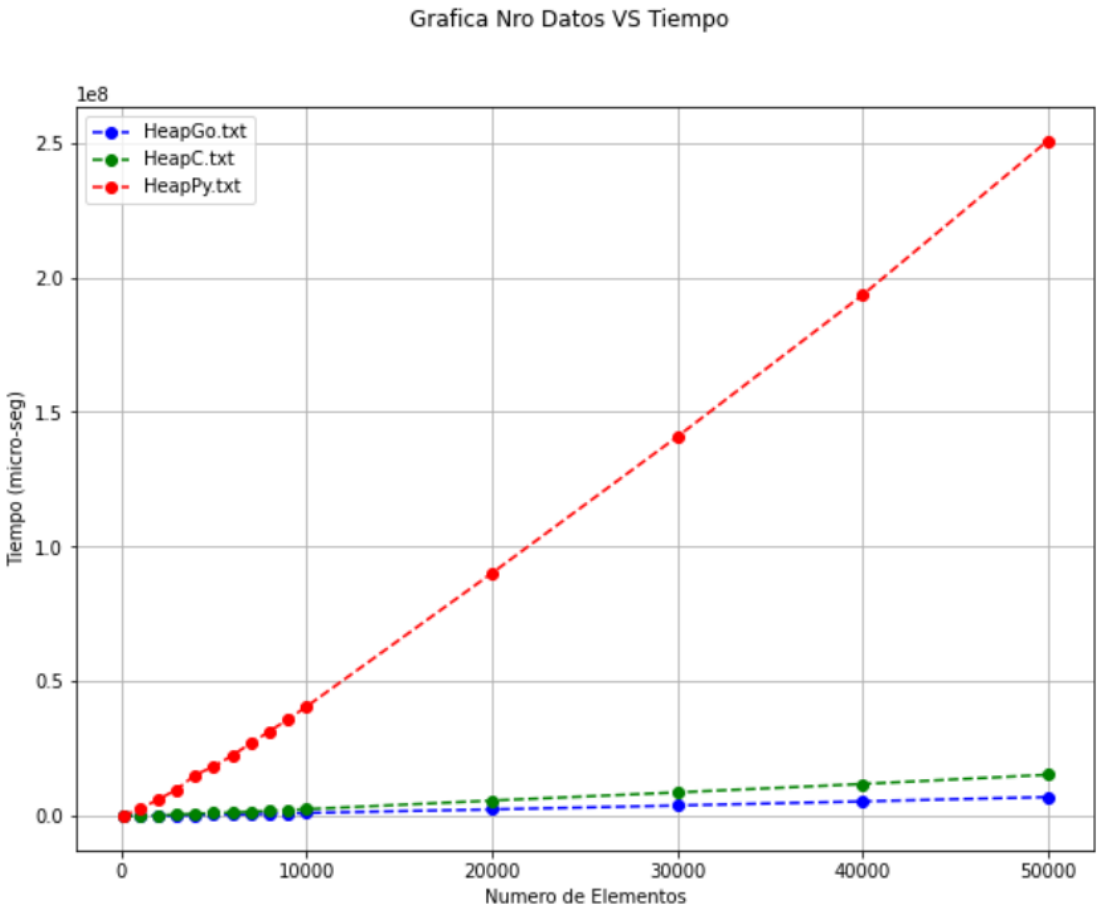


Figura: Comparación de *Heap Sort*

Heap Sort: Limite Superior y Limite Inferior

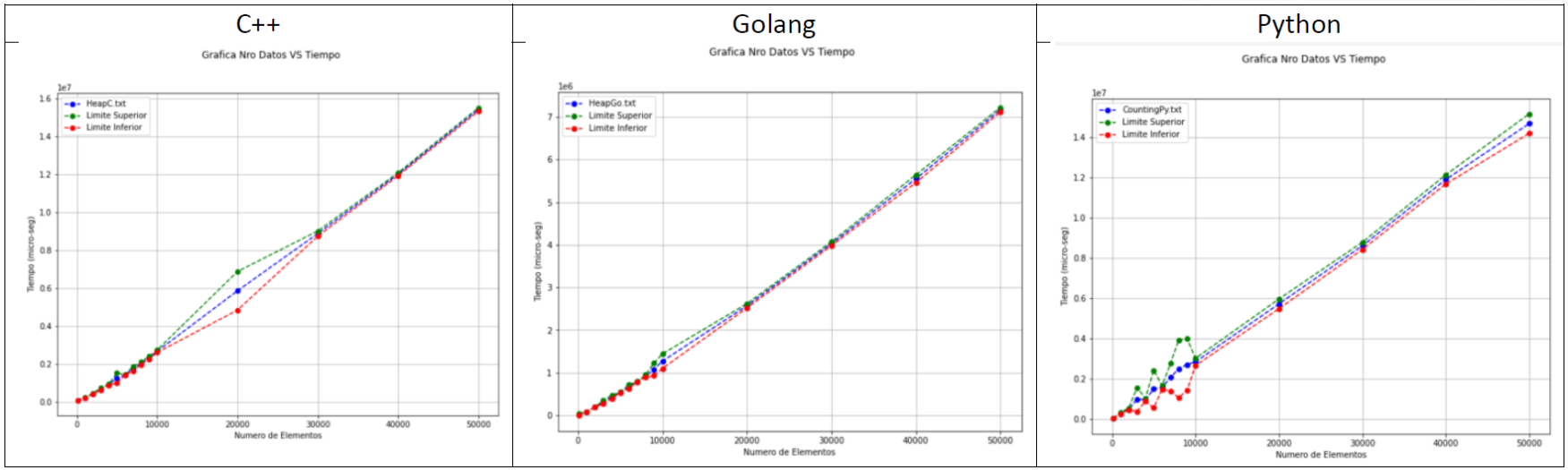


Figura: *Heap Sort*: Limite Superior y Limite Inferior

Implementación

La implementación de la práctica se encuentra en el siguiente repositorio:

`https://github.com/HaydeLuzmilaHc/Grupo-01-Practicas-de-AyED`

Resultados

Para el reporte de resultados se han usado gráficos variados, dentro de los cuales tenemos dos tipos:

1. Comparaciones de Algoritmos: En estos gráficos se ha comparados los 4 algoritmos en 1 sola gráfica, para poder identificar el mas rápido.
2. Tiempo Promedio Límites Superiores e Inferiores, para el estudio del límite superior, caso promedio y límite superior, se ha considerado como promedio el promedio de los 5 casos, el límite superior se ha considerado como la media mas 2 desviaciones estándar, el límite inferior es la media menos 2 desviaciones estándar, esto se realizo para cada uno de los algoritmos.

Resultados: Gráfica del Algoritmos en C++

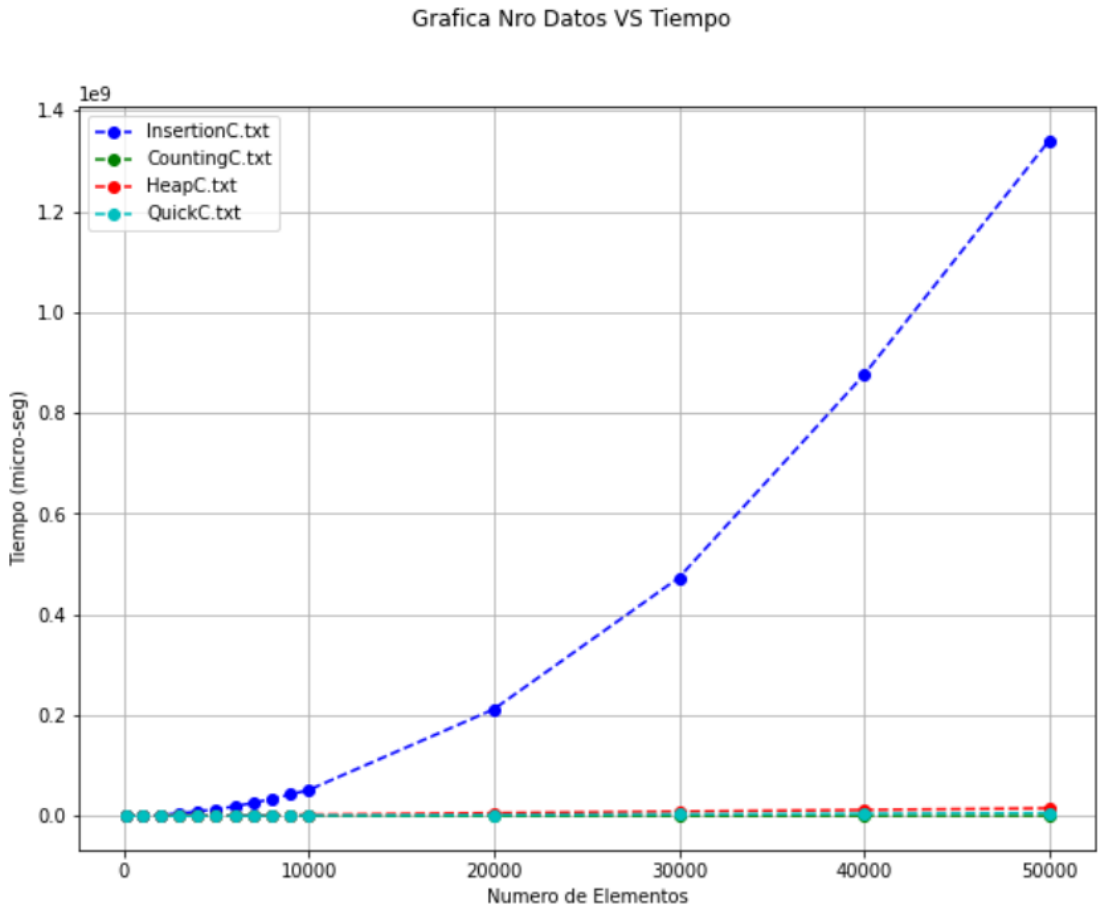


Figura: Comparación de Algoritmos en C++

Resultados: Gráfica del Algoritmos en Python

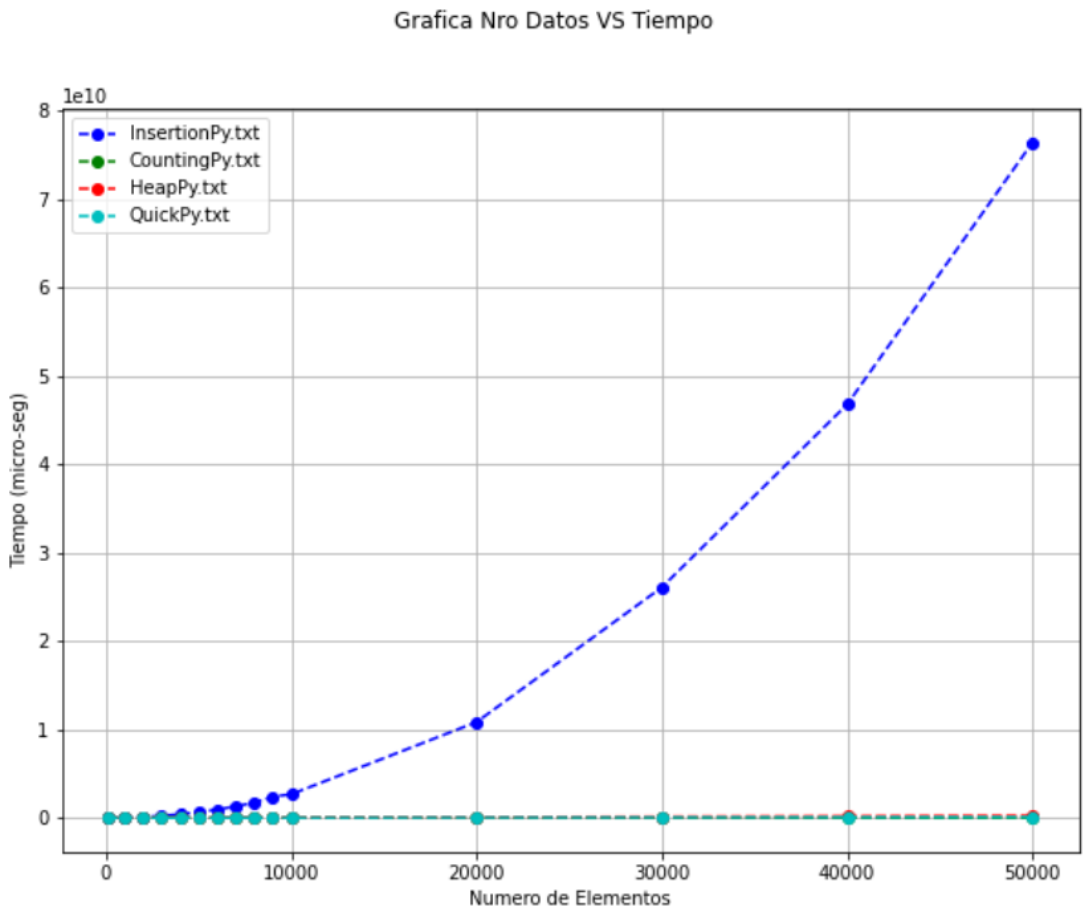


Figura: Comparación de Algoritmos en Python

Resultados: Gráfica del Algoritmos en Golang

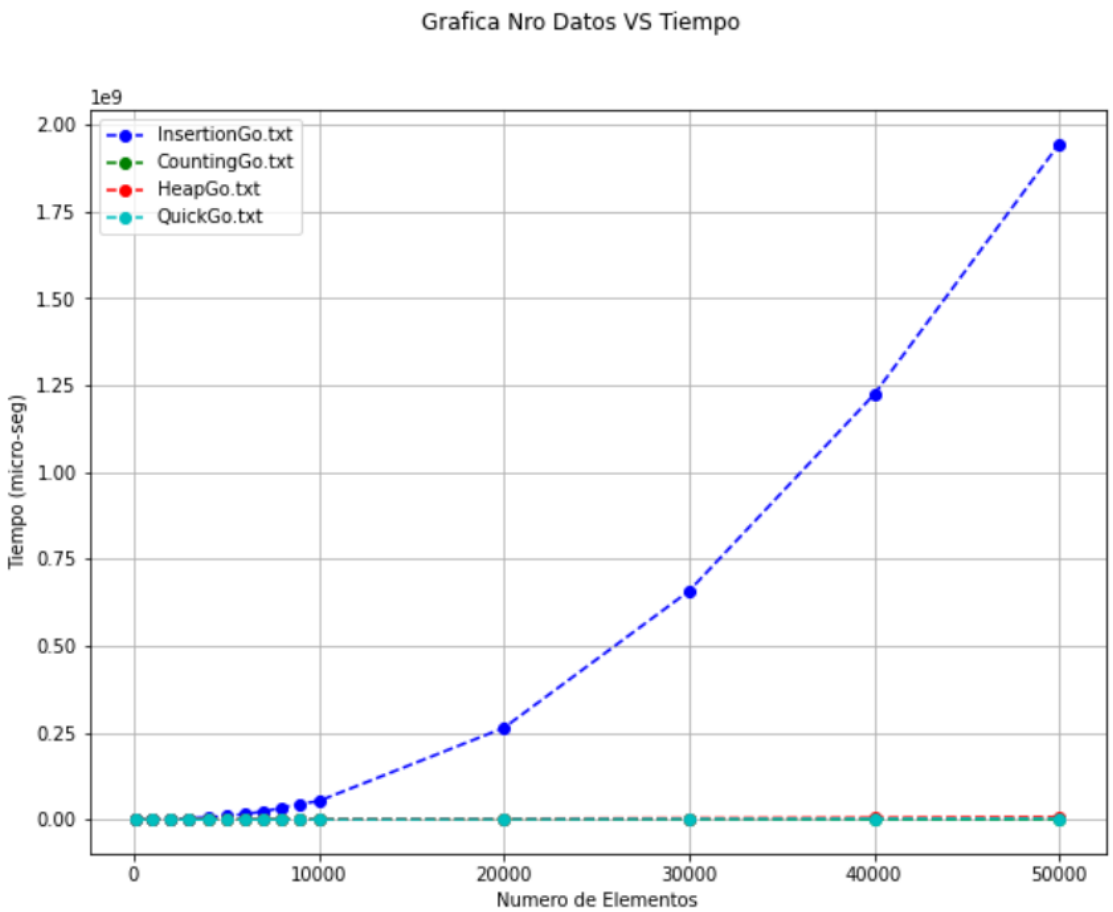


Figura: Comparación de Algoritmos en Golang

Resultados: Tablas Comparativa de Tiempos

Los tiempos que se muestran en las tablas son el promedio de 5 muestras con cada tamaños de entrada formateadas en nanosegundos.

Tiempos de ejecución: *Counting Sort*

Tamaño	Lenguaje Golang	Lenguaje C++	Lenguaje Python
100	4156.4	6411.2	28610.2
1000	13108.6	16571.4	277566.9
2000	17744.0	26460.4	493144.9
3000	29865.4	51471.6	968122.4
4000	116422.8	51320.6	954723.3
5000	30517.2	64278.6	1488065.7
6000	38804.8	78167.0	1567077.6
7000	46604.0	91283.8	2057838.4
8000	51361.0	104795.4	2499723.4
9000	73105.8	121225.4	2708625.7
10000	67685.6	131594.4	2835369.1
20000	175157.4	278895.0	5719041.8
30000	358891.8	426290.2	8603239.0
40000	465343.2	594650.0	11903095.2
50000	588081.2	756761.2	14678049.0

Cuadro: Tiempos de ejecución del algoritmo *Counting Sort*

Tiempos de ejecución: *Quick Sort*

Tamaño	Lenguaje Golang	Lenguaje C++	Lenguaje Python
100	3393.8	29545.6	146007.5
1000	41710.2	80245.0	1518249.5
2000	86745.4	160182.8	2728891.3
3000	147516.6	255427.0	4216384.8
4000	190250.4	353236.2	5927562.7
5000	256520.0	458179.0	7825803.7
6000	298074.8	586476.2	9375333.7
7000	349201.4	658908.2	10919904.7
8000	409211.6	762335.6	13009119.0
9000	503380.0	871668.4	15661668.7
10000	519730.0	993605.2	17081308.3
20000	1107891.6	2171292.4	42290687.5
30000	1712465.8	3415140.6	56382703.7
40000	2355919.2	4624127.8	77776670.4
50000	3043727.2	5934884.6	98295640.9

Cuadro: Tiempos de ejecución del algoritmo *Quick Sort*

Tiempos de ejecución: *Insertion Sort*

Tamaño	Lenguaje Golang	Lenguaje C++	Lenguaje Python
100	34279.6	9664.8	238466.2
1000	1047769.2	560293.2	24469757.0
2000	1618634.6	2146801.2	101745653.1
3000	3545198.0	4863337.4	231694602.9
4000	6452998.0	8871870.6	419051647.1
5000	10572702.4	13384660.0	648374462.1
6000	15932791.8	19018828.8	952532958.9
7000	23171703.6	26055904.2	1307987546.9
8000	34123140.8	33577623.6	1729411411.2
9000	44976422.2	42766146.8	2414094448.0
10000	53407743.2	51611769.6	2681028413.7
20000	264132010.6	211818186.2	10851650571.8
30000	657936185.8	473043358.2	26091095590.5
40000	1223976990.6	875843928.2	46787051725.3
50000	1945301778.2	1340750614.6	76417606306.0

Cuadro: Tiempos de ejecución del algoritmo *Insertion Sort*

Tiempos de ejecución: *Heap Sort*

Tamaño	Lenguaje Golang	Lenguaje C++	Lenguaje Python
100	17632.2	55352.4	199508.6
1000	87206.6	195362.0	2769327.1
2000	189115.2	412048.0	6170511.2
3000	315177.6	672130.4	9947490.6
4000	431417.6	904263.6	15331888.1
5000	539877.0	1239145.0	18486928.9
6000	673315.6	1413743.6	22559595.1
7000	787593.2	1730729.4	27085971.8
8000	922658.4	2028979.8	31501817.7
9000	1079712.4	2321449.8	35928535.4
10000	1272634.2	2656174.8	40650987.6
20000	2569350.4	5853268.2	90158414.8
30000	4024258.2	8876363.8	140787458.4
40000	5548282.2	12012330.8	193399429.3
50000	7164878.0	15432487.0	250664281.8

Cuadro: Tiempos de ejecución del algoritmo *Heap Sort*

Conclusiones

1. La implementación de todos los algoritmos en el lenguaje de C++ tiene mejores resultados en comparación con los algoritmos de Python y Goland.
2. Los gráfico del tiempo de procesamiento de cada uno de los algoritmos se rigen en función matemática que representa su complejidad algorítmica.
3. El tiempo de procesamiento va depender mucho del tamaño de data que ingresemos.
4. Las características propias del lenguaje influyen directamente en el tiempo de respuesta del algoritmo.

Algoritmos y Estructura de Datos (AyED)
Práctica 01: Algoritmos de Ordenamiento

Estudiantes:
Franklin Canaza Ccori
Hayde Luzmila Humpire Cutipa
Jair Francesco Huaman Canqui
Jhoel Salomon Tapara Quispe

Docente: Dr. Vicente Machaca Arceda

Escuela Profesional de Ciencias de la Computación
Universidad Nacional de San Agustín